Implementation

The three sizes were 50, 151, and 450. I chose these sizes because they gave a wide range while also all fitting well onto a graph together. I was originally going to do magnitudes of 10, like 10, 100 and 1000, but this resulted in the graphs being dominated by the hash table of size 1000, so I decided to make tables with somewhat similar sizes.

The number of collisions started at zero and was incremented whenever there was an insertion that attempted to access a non-empty index. For separate chaining, if the linked list at the calculated index was non-empty, that was a collision, no matter its size. For open addressing, if the original index was non-empty, then that was one collision, and the probing that occurred after did not change the number of collisions. The load factor was recalculated after each insertion, and then the collision and load factor were loaded into a csv so that their relationship could be graphed in Excel.

In order to test each implementation, I created two hash table objects. HashTableSC uses a separate chaining collision resolution scheme while HashTableOA uses an open addressing collision resolution scheme. Each of these objects has two different hash function insertions, one being key mod table size and the other being mid square. Thus for one run of my program, I create 4 different hash table objects, two of each collision resolution scheme. I then run one for loop from 0 to table size, and insert the same number into each of the different hash tables such that each combination of collision resolution/hash function is used. After each insertion, I load the current number of collisions and load factor into the designated file for each combination.

Graphs of Data

Series 1 = table size of 50 , Series 2 = table size of 151, Series 3 = table size of 450

Discussion of Graphs

The most effective implementation at all table sizes was the separate chaining collision resolution scheme with a key mod table size hashing function. Separate chaining produces less collisions compared to open addressing because after a collision, the amount of empty buckets remains the same. The list size at that particular index increases, but that doesn’t decrease the number of non-empty buckets, so the chance of a future collision remains the same as it was before the insertion. The chance of a collision only increases when an empty bucket becomes non-empty, thus when using separate chaining it is only an insertion that does not cause a collision that will increase the chances of a future collision. However, open addressing will always result in an empty bucket becoming non-empty, as if the index it attempts to access is full it uses linear probing to find a different, empty index. Thus when using the open addressing collision resolution scheme, an insertion will always increase the chance of a future collision.

Interestingly, the key mod table size hashing function distributed the keys more evenly than the mid square hashing function. Although the mid square hashing function is significantly more complicated, it failed to produce a more random distribution of index numbers. In fact, the difference was so significant that the open addressing key mod table produced less collisions compared to the separate chaining mid square hash table. This is because “mid square is designed for applications where the keys aren't really random, and mod table size would produce a lot of collisions” – Cheryl Resch. The exact same numbers were inserted for every table of the same size, so the difference in collisions was solely due to the implementation of the hash table. Even though open addressing is a less effective collision resolution scheme, it can still result in less collisions than a chain hash table if the hashing function itself distributes the keys more uniformly.

One of the most noticeable differences between the graphs is their smoothness, especially at smaller sizes. The hashing implementations with less collisions have noticeably more uneven slopes. The most effective implementation, the separate chaining with a key mod table size hash function, has a very jagged graph. This is because when no collision occurs the open addressing with a mid square hash function has a much smoother graph, and this is because there is a collision after nearly every insertion after the load factor goes above 0.6. This is the result of open addressing having 60% of its buckets be non-empty at this load factor, while with separate chaining the load factor could be 0.6 but the percentage of non-empty buckets will be less than or equal to 60%.

The ratio between the number of collisions and table size remained relatively constant for each collision resolution scheme and hashing function combination. Thus the total number of collisions increased linearly with the table size. This makes sense because the chance of a collision happening is based on the number of empty buckets compared to non-empty buckets, not the size of the table itself. Thus the shape of the graphs remains relatively constant across all sizes, with their relative jaggedness/straightness remaining the same at each load factor.